

# Church Machine – Ti60 F225 Hardware Guide

## Efinix Ti60 F225 – Church Machine Hardware Reference

v1.0 – 2026-04-29 CONFIDENTIAL

### Board specifications

Feature	Value
<b>Device</b>	Efinix Titanium EFT90A
<b>Board</b>	Sipeed Ti60 F225 Development Kit
<b>Process</b>	90 nm
<b>Clock</b>	50 MHz on-board crystal (pin B8)
<b>LEDs</b>	4 × active-HIGH (USER_LED[3:0])
<b>Button</b>	1 × active-LOW USER_PB (external pull-up on board)
<b>UART bridge</b>	FTDI FT232H USB-UART (115200 baud, USB-C connector J4)
<b>Memory</b>	256 KB SRAM via Efinix EBR tiles (2 × 128 KB blocks)
<b>I/O banks</b>	Bank 3 (LVCMOS33)
<b>Synthesis toolchain</b>	Efinity IDE (official Efinix toolchain)

# Toolchain installation

## 1. Efinity IDE (required for synthesis and programming)

Efinity IDE is the official Efinix toolchain; it cannot be replaced by yosys/nextpnr for production synthesis on EFT90A silicon.

1. Download from <https://www.efinixinc.com/efinity-ide>
2. Run the installer on your local machine (Linux / Windows / macOS)
3. Register a licence — a 30-day evaluation licence is available at no cost
4. Add the Efinity bin/ directory to your shell PATH:

```
export PATH="/path/to/efinity/bin:$PATH"
```

5. Verify:

```
efinity --version
```

## ChromeOS / Crostini (Linux container on Chromebook)

Efinity crashes on startup under Crostini due to GPU/OpenGL unavailability. Launch with X11 platform forced and software rendering:

```
QT_QPA_PLATFORM=xcb LIBGL_ALWAYS_SOFTWARE=1 /home/$USER/efinity/  
2025.2/bin/efinity &
```

Create a permanent launcher so you never have to remember the flags:

```
cat > ~/efinity.sh << 'EOF'  
#!/bin/bash  
QT_QPA_PLATFORM=xcb LIBGL_ALWAYS_SOFTWARE=1 /home/$USER/efinity/  
2025.2/bin/efinity "$@"  
EOF  
chmod +x ~/efinity.sh
```

Then run ~/efinity.sh to open it. The serial port blocking the FPGA Connect button is Efinity's built-in serial terminal — close it (Tools menu) before switching back to the Church Machine IDE.

## 2. openFPGALoader (for CLI flashing)

```
# Ubuntu / Debian  
sudo apt install openfpgaloader
```

```
# Arch Linux
sudo pacman -S openfpgaloader
```

```
# From source
git clone https://github.com/trabucayre/openFPGALoader
cd openFPGALoader && cmake . && make && sudo make install
```

### 3. Serial console tools

```
# Ubuntu / Debian
sudo apt install picocom minicom
```

```
# Arch Linux
sudo pacman -S picocom
```

Add your user to the dialout group so you can access /dev/ttyUSB0 without sudo:

```
sudo usermod -aG dialout $USER
# log out and back in for this to take effect
```

---

## RTL generation

Generate build/church\_ti60\_f225.il (Amaranth RTLIL) and build/church\_ti60\_f225.v (plain Verilog for Efinity) in one step:

```
python3 -m hardware.gen_rtlil build --ti60
```

Expected output:

```
Generated: build/church_ti60_f225.il
  File size: 2,071,590 bytes
  Lines: 73,326
  Verilog: build/church_ti60_f225.v
  Fixed 1 \macc cell(s) → behavioural Verilog
```

The generator pipeline is: 1. **Amaranth** → **RTLIL** via `amaranth.back.rtlil.convert` 2. **Yosys** `proc; flatten; alumacc; clean; write_verilog -noattr` — lowers Amaranth primitives and merges adder trees 3. **\$macc post-processor** (`hardware/gen_rtlil.py: _fix_macc_cells`) — Yosys's `alumacc` pass folds constant-coefficient multiplies into `\macc` cells that `write_verilog` cannot emit as plain operators and that Efinity rejects. The post-processor detects these cells (`B_WIDTH=0`, leading constant in A) and replaces each with a behavioural assign statement (`Y_signal = A_signal * constant`), which Efinity synthesises using its built-in multiplier primitives.

The output Verilog contains **zero** \smacc **instantiations** and imports cleanly into Efinity 2025.2.

Quick module-load check:

```
python3 -c "from hardware.ti60_f225 import ChurchTi60F225;
           print('OK')"
```

---

## Build and flash

All Efinity commands must run on your local machine (Efinity IDE is not available in Replit).

### Step 1 — Synthesis

```
efinity -t build/church_ti60_f225.rtlil -d EFT90A -p pinout.csv
        -o build/church_ti60_f225.edf
```

Expected output: build/church\_ti60\_f225.edf (~500 KB Edif netlist, no errors)

Timing target: **50 MHz** (20 ns period). If synthesis fails to close timing, try lowering the target to 25 MHz:

```
efinity -t build/church_ti60_f225.rtlil -d EFT90A --freq 25 -p
        pinout.csv -o build/church_ti60_f225.edf
```

### Step 2 — Place and route

```
efinity --pnr build/church_ti60_f225.edf -p pinout.csv -o build/
        church_ti60_f225_pnr.edf
```

Expected output: build/church\_ti60\_f225\_pnr.edf + timing report.

### Step 3 — Bitstream

```
efinity --bitstream build/church_ti60_f225_pnr.edf -o build/
        church_ti60_f225.fs
```

Expected output: build/church\_ti60\_f225.fs (~1-2 MB binary bitstream)

### Step 4 — Flash

#### CLI (openFPGALoader):

```
openFPGALoader -b ti60f225 build/church_ti60_f225.fs
```

## GUI (Efinity Programmer):

```
efinity --program build/church_ti60_f225.fs
```

Troubleshooting: - `lsusb | grep -i efinix` — confirm the FTDI bridge is enumerated - If programming times out, try `openFPGALoader -b ti60f225 --verbose build/church_ti60_f225.fs`

---

## UART serial console

Connect USB-C to J4. The FTDI FT232H bridge enumerates as `/dev/ttyUSB0` (Linux) or `COM<N>` (Windows).

```
picocom -b 115200 /dev/ttyUSB0  
# or  
minicom -D /dev/ttyUSB0 -b 115200
```

Protocol: 8N1, 115200 baud, no flow control.

On boot, the firmware prints:

```
CHURCH Ti60 v1.0  
<NIA as hex>HALT
```

On each button press (single-step):

```
S:<NIA as hex>HALT
```

On fault:

```
S:<NIA as hex>F:<fault code as hex>HALT
```

---

## Boot sequence

1. 16-cycle boot delay on power-up
  2. Boot ROM initialises CR6 (c-list), CR14 (CLOOMC/code), CR8, CR15
  3. Boot namespace (NS) and c-list entries written to EBR at init time
  4. 3-second startup delay (hardware) → UART banner sent
  5. Machine enters HALTED state; press USER\_PB to single-step
-

## MMIO register map

All IO devices are mapped at 0x40000000 (address bit 30 set, bit 31 clear). The MMIO register selector is addr[5:2] (4-bit word index within the MMIO range).

Sel	Address	Name	Dir	Boot NS slot	CRC seal
0	0x40000000	LED[0]	R/ W	7	0x366A
1	0x40000004	LED[1]	R/ W	7	—
2	0x40000008	LED[2]	R/ W	7	—
3	0x4000000C	LED[3]	R/ W	7	—
4	0x40000010	LED[4]	R/ W	7	—
5	0x40000014	UART_TX	W	8	0x43A4
6	0x40000018	UART_STATUS	R	8	—
7	0x4000001C	UART_RX	R	8	—
8- 9	—	(reserved)	—	—	—
10	0x40000028	BTN	R	9	0x0F00
11	0x4000002C	TIMER.TICKS_LO	R	10	0xEBC6
12	0x40000030	TIMER.TICKS_HI	R	10	—
13	0x40000034	TIMER.TOD_EPOCH	R/ W	10	—
14	0x40000038	TIMER.ALARM_CMP	R/ W	10	—
15	0x4000003C	TIMER.ALARM_CTL	R/ W	10	—

CRC seals shown are the NS entry word2\_w3 values (CRC-16/CCITT over GT[24:0] + location + word2).

## IO devices

### LED (Boot NS Slot 7)

**Identity:**

Property	Value
MMIO base	0x40000000
Words	5 (offsets 0-4, one per LED channel)
GT type	GT_TYPE_INFORM
Permissions	R W
b_flag	1
GT word 0	0x86800007
CRC seal	0x366A

### Register map:

Offset	Address	Name	Bits	Ti60 F225 pin	Active
0	0x40000000	LED[0]	[2:0]={B,G,R}	led0	HIGH
1	0x40000004	LED[1]	[2:0]={B,G,R}	led1	HIGH
2	0x40000008	LED[2]	[2:0]={B,G,R}	led2	HIGH
3	0x4000000C	LED[3]	[2:0]={B,G,R}	led3	HIGH
4	0x40000010	LED[4]	[2:0]={B,G,R}	(no pin)	—

Only bit 0 (R) drives a physical pin. The Ti60 F225 has 4 physical LEDs; offset 4 is a register-only placeholder with no pin. Bits [31:3] ignored on write, read as zero.

### Usage:

DWRITE DR\_src, [CR\_led + N] ; N = 0..4, DR\_src[0] = R (1 = LED on)

DREAD DR\_dst, [CR\_led + N] ; read back register value

### Pre-boot LED meanings (hardware status display):

LED	Pre-boot meaning
led0	ON while booting, OFF when boot complete
led1	ON when running; blinks at 1 Hz when halted
led2	ON when fault detected
led3	ON until boot complete

Post-boot: software controls all four LEDs via DWRITE to offsets 0-3.

## UART (Boot NS Slot 8)

### Identity:

Property	Value
MMIO base	0x40000014
Words	3 (offsets 0-2)
GT type	GT_TYPE_INFORM
Permissions	R W
b_flag	1
GT word 0	0x86800008
CRC seal	0x43A4
Physical bridge	FTDI FT232H, 115200 baud, USB-C J4

### Register map:

Offset	Address	Name	Dir	Meaning
0	0x40000014	TX	W	Byte to transmit ([7:0]); send when idle
1	0x40000018	STATUS	R	[0] = TX ready (1=idle, 0=busy); [31:1]=0
2	0x4000001C	RX	R	Received byte ([7:0]); 0x00 = empty

### Usage:

```

; Poll until ready, then send byte 'A'
uart_wait:
    DREAD  DR1, [CR_uart + 1]
    ANDI   DR1, DR1, #1
    BEQ    uart_wait
    MOVI   DR1, #0x41          ; 'A'
    DWRITE DR1, [CR_uart + 0]

; Receive byte
    DREAD  DR1, [CR_uart + 2]

```

The MMIO TX path shares the physical UART with the debug FSM (banner/halt/step/fault messages). The debug FSM takes priority; MMIO TX sends when the debug FSM is not busy.

### Attenuation:

Attenuated GT	Perms	Use case
TX-only	W	

<b>Attenuated GT</b>	<b>Perms</b>	<b>Use case</b>
		Send-only thread (no status poll or RX)
RX+STATUS	R	Receive-only thread
Full	R W	Full UART access

## BTN (Boot NS Slot 9)

### Identity:

<b>Property</b>	<b>Value</b>
MMIO base	0x40000028
Words	1 (offset 0 only)
GT type	GT_TYPE_INFORM
Permissions	R
b_flag	1
GT word 0	0x82800009
CRC seal	0x0F00
Physical button	USER_PB (active-LOW, external pull-up)

### Register map:

<b>Offset</b>	<b>Address</b>	<b>Name</b>	<b>Dir</b>	<b>Meaning</b>
0	0x40000028	BTN	R	[0] = pressed (1=pressed); [31:1]=0

The hardware normalises the active-LOW signal so bit 0 is always 1 when the button is held down, regardless of the physical polarity. The debouncer is a 3-stage synchroniser; the register reflects the debounced level (not a pulse).

### Edge-detect pattern:

btn\_poll:

```
DREAD DR1, [CR_btn + 0] ; current state
; compare DR1[0] with saved DR2[0]
; rising edge (press) = DR1[0]==1 and DR2[0]==0
; falling edge (release) = DR1[0]==0 and DR2[0]==1
; save DR1 → DR2 for next iteration
```

DWRITE against this GT faults with PERMISSION — no W permission is granted.

## TIMER (Boot NS Slot 10)

### Identity:

Property	Value
MMIO base	0x4000002C
Words	5 (offsets 0-4)
GT type	GT_TYPE_INFORM
Permissions	R W
b_flag	1
GT word 0	0x8680000A
CRC seal	0xEBC6
Clock rate	50 MHz (20 ns tick)
32-bit TICKS_LO wrap	~85.9 s

### Register map:

Offset	Address	Name	Dir	Meaning
0	0x4000002C	TICKS_LO	R	Low 32 bits of 64-bit free-running tick counter
1	0x40000030	TICKS_HI	R	High 32 bits of 64-bit tick counter
2	0x40000034	TOD_EPOCH	R/W	Unix time in seconds (set by boot or IDE)
3	0x40000038	ALARM_CMP	R/W	Alarm compare value (matched against TICKS_LO)
4	0x4000003C	ALARM_CTL	R/W	[0]=armed, [1]=fired (write 1 to bit 1 to clear)

### Current time formula:

$$\text{current\_unix} = \text{TOD\_EPOCH} + (\text{TICKS\_LO\_now} - \text{TICKS\_LO\_at\_boot}) / 50\_000\_000$$

### Elapsed-time pattern:

```

DREAD DR1, [CR_timer + 0] ; TICKS_LO start
DREAD DR2, [CR_timer + 1] ; TICKS_HI start
; ... work ...
DREAD DR3, [CR_timer + 0] ; TICKS_LO end
DREAD DR4, [CR_timer + 1] ; TICKS_HI end
; elapsed = DR3 - DR1 (32-bit, handles wrap)

```

### Alarm pattern:

```

DREAD DR1, [CR_timer + 0] ; read current TICKS_LO
ADDI DR1, DR1, #delay_ticks ; target = now + delay
DWRITE DR1, [CR_timer + 3] ; set ALARM_CMP
MOVI DR2, #0x01
DWRITE DR2, [CR_timer + 4] ; arm
alarm_poll:
  DREAD DR2, [CR_timer + 4]
  ANDI DR2, DR2, #0x02 ; test fired bit
  BEQ alarm_poll
MOVI DR3, #0x02
DWRITE DR3, [CR_timer + 4] ; clear fired flag

```

## Additional IO port options

The Ti60 F225 Development Kit exposes the following GPIO banks for expansion:

Bank	Standard	Notes
Bank 3	LVC MOS33	Used for UART, LEDs, button; additional pins available
J5 / J6 expansion headers	LVC MOS33	Free GPIO; assign in Efinity constraint file
JTAG header	—	On-board JTAG for Efinity Programmer

To assign an expansion GPIO: 1. Add an IO\_LOC and IO\_PORT entry to your Efinity .pdc pin constraints file 2. Add the corresponding Signal to ChurchTi60F225 and connect it in elaborate() 3. Re-synthesise, place-and-route, regenerate bitstream, and reflash

Available peripheral candidates for future integration: - SPI flash (on-board, shared with bitstream; careful with CE timing) - I2C sensors via expansion header - HDMI differential pairs (available on Ti60 silicon; not currently routed in the design) - Additional UART channels via Bank 3 free pins

# Troubleshooting

Symptom	Likely cause	Fix
RTL generation fails	Import error in hardware/	Run module-load check (see above)
Synthesis timing fails at 50 MHz	Long combinational path	Lower target to 25 MHz or pipeline critical paths
openFPGALoader device not found	USB not enumerated	<code>lsusb   grep -i efinix;</code> check cable and driver
UART outputs nothing	TX/RX swapped, or wrong baud	Verify <code>uart_tx/uart_rx</code> pins in constraint file; confirm 115200
LEDs don't respond post-boot	DWRITE offset wrong	Confirm offset 0-3 for Ti60; offset 4 has no physical pin
Button read always 0	Debounce sync	Hold button down; the register is level (not pulse)